4

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

| APPLICATION NO. | FILING DATE | FIRST NAMED INVENTOR | ATTORNEY DOCKET NO. | CONFIRMATION NO. |
|---|---|---|---|---|
| 10/706,127 | 11/12/2003 | Maurizio Spadari | 03-1868 1496.00348 | 6216 |

24319 7590 06/05/2007

LSI LOGIC CORPORATION
1621 BARBER LANE
MS: D-106
MILPITAS, CA 95035

| EXAMINER | |
|---|---|
| CHU, GABRIEL L | |

| ART UNIT | PAPER NUMBER |
|---|---|
| 2114 | |

| MAIL DATE | DELIVERY MODE |
|---|---|
| 06/05/2007 | PAPER |

**Please find below and/or attached an Office communication concerning this application or proceeding.**

The time period for reply, if any, is set in the attached communication.

| Office Action Summary | Application No. | Applicant(s) | |
|---|---|---|---|
| | 10/706,127 | SPADARI, MAURIZIO | |
| | Examiner | Art Unit | |
| | Gabriel L. Chu | 2114 | |

*-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --*

**Period for Reply**

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

**Status**

1)☒ Responsive to communication(s) filed on *12 November 2003*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

**Disposition of Claims**

4)☒ Claim(s) *1-20* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-20* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

**Application Papers**

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *12 November 2003* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

**Priority under 35 U.S.C. § 119**

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

**Attachment(s)**

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

## DETAILED ACTION

### *Claim Rejections - 35 USC § 102*

1. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

**2. Claims 1-4, 8, 9, 20 rejected under 35 U.S.C. 102(e) as being anticipated by US 6769116 to Sexton.**

3. Referring to claim 1, Sexton discloses an apparatus comprising: an analysis block configured to generate debug information in response to (i) a command input (From line 15 of column 8, "To debug memory corruption, a predetermined pointer is set to the desired address, which is checked in the instrumented memory management routines, and a breakpoint is set on the breakpointable function. Thus, the execution of large portions of the program can be quickly passed over before starting the intensive watchpointing or other memory inspection procedure."), (ii) one or more simulation outputs (From line 43 of column 8 (with emphasis), "Preferably, the instrumented code is conditionally compiled into the program. Thus, the **development version** of the program would include the instrumented code, but the production version of the program need not include the instrumented code."), and (iii) one or more compiler outputs (From line 43 of column 8 (with emphasis), "Preferably, the instrumented code

is conditionally **compiled** into the program. Thus, the development version of the program would include the instrumented code, but the production version of the program need not include the instrumented code.");

a graphical user interface (From line 55 of column 5, "Computer system 100 may be coupled via bus 102 to a display 112, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 114, including alphanumeric and other keys, is coupled to bus 102 for communicating information and command selections to processor 104. Another type of user input device is cursor control 116, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 104 and for controlling cursor movement on display 112. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane." From line 55 of column 8, "When the instrumented code have been compiled into memory management or other library routines of the program under a debugging switch, the programmer is now ready to diagnose the cause of the memory corruption. Preferably, the following steps are performed within a symbolic debugger such as sdb (a standard UNIX.TM. debugger) or gdb (a freeware debugger), but some of the steps may optionally be performed by hard coding some additional code into the program and recompiling.") configured (i) to present said command input in response to one or more user input parameters and (ii) to display said debug information (From line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function

that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." From line 10 of column 9, "When the breakpoint is triggered (e.g. when a memory management routine handles an object with the address set in the global debug pointer), the debugger breaks execution of the problem, allowing the programmer to issue debugging commands to inspect memory locations and continue the program's running from the breakpoint. For example, when the memory allocation routine allocates memory for an object that starts with the address set in the global debug pointer, then the debugging routine would be called, triggering the breakpoint and suspending execution of the program within the debugger."); and

a memory circuit configured to store said one or more simulation outputs and said one or more compiler outputs (From column 5, RAM, ROM. From column 6, computer-readable medium. From column 7, virtual memory, secondary storage.).

4. Referring to claim 2, Sexton discloses said one or more user input parameters comprise identification of a memory address (From line 15 of column 8, "To debug memory corruption; a predetermined pointer is set to the desired address, which is checked in the instrumented memory management routines, and a breakpoint is set on the breakpointable function. Thus, the execution of large portions of the program can be

quickly passed over before starting the intensive watchpointing or other memory inspection procedure.").

5. Referring to claim 3, Sexton discloses said one or more user input parameters comprise identification of specific memory accesses (From line 30 of column 9, "At block 208, e.g. after the last breakpoint is triggered, a watchpoint is set to detect changes to the memory location that is being corrupted. Alternatively, the memory location can be manually inspected by single stepping through the program. After setting the watchpoint, execution of the program is continued until the watchpoint is triggered, which occurs upon a change to the corrupted memory location, thereby breaking execution of the program. At each of the breaks in execution due to encountering the watchpoints, the programmer can use the debugger to inspect the program and data in search of the cause of the memory corruption. If the programmer does indeed concluded that the latest watchpoint detected the actual corruption, rather than an intended use of the memory location, then the programmer can tell from the current program counter in the debugger which statement caused the corruption.").

6. Referring to claim 4, Sexton discloses said one or more user input parameters comprise one or more of a command for expanding the display of a memory map, a command for retrieving information related to accesses to said memory map, a command for retrieving assembly code related to particular accesses and one or more commands related to filtering operations (At least, from line 30 of column 9, "At block 208, e.g. after the last breakpoint is triggered, a watchpoint is set to detect changes to the memory location that is being corrupted. Alternatively, the memory location can be

manually inspected by single stepping through the program. After setting the watchpoint, execution of the program is continued until the watchpoint is triggered, which occurs upon a change to the corrupted memory location, thereby breaking execution of the program. At each of the breaks in execution due to encountering the watchpoints, the programmer can use the debugger to inspect the program and data in search of the cause of the memory corruption. If the programmer does indeed concluded that the latest watchpoint detected the actual corruption, rather than an intended use of the memory location, then the programmer can tell from the current program counter in the debugger which statement caused the corruption.").

7. Referring to claim 8, Sexton discloses said one or more user input parameters are entered using one or more of a mouse, a keyboard, a touch screen and voice recognition (From line 55 of column 5, "Computer system 100 may be coupled via bus 102 to a display 112, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 114, including alphanumeric and other keys, is coupled to bus 102 for communicating information and command selections to processor 104. Another type of user input device is cursor control 116, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 104 and for controlling cursor movement on display 112. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane." ).

8. Referring to claim 9, Sexton discloses said graphic user interface is configured to provide an indication of receipt of said user input parameters (Wherein, at least, in

Sexton, **something happens** as a result of the programmer graphically using the interface, e.g., the invention of Sexton.).

9. Referring to claim 20, Sexton discloses means for identifying a memory address in a user defined memory map (From line 9 of column 4, "To use these instrumented memory management routines, a programmer ascertains the address of the object that was corrupted, typically by inspecting the core dump, and sets the global pointer to that address." From line 15 of column 8, "To debug memory corruption, a predetermined pointer is set to the desired address, which is checked in the instrumented memory management routines, and a breakpoint is set on the breakpointable function. Thus, the execution of large portions of the program can be quickly passed over before starting the intensive watchpointing or other memory inspection procedure." Wherein the user is the programmer, the memory mapping is programmed, directly or indirectly, by the programmer.);

means for retrieving one or more accesses related to an identified memory address from simulation results; (C) identifying a specific one of said one or more accesses to be examined (From line 10 of column 9, "When the breakpoint is triggered (e.g. when a memory management routine handles an object with the address set in the global debug pointer), the debugger breaks execution of the problem, allowing the programmer to issue debugging commands to inspect memory locations and continue the program's running from the breakpoint. For example, when the memory allocation routine allocates memory for an object that starts with the address set in the global debug pointer, then the debugging routine would be called, triggering the breakpoint

and suspending execution of the program within the debugger." From line 43 of column 8 (with emphasis), "Preferably, the instrumented code is conditionally compiled into the program. Thus, the **development version** of the program would include the instrumented code, but the production version of the program need not include the instrumented code."); and

means for retrieving one or more types of debug information related to said specific access from said simulation results (From line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." From line 30 of column 9, "At block 208, e.g. after the last breakpoint is triggered, a watchpoint is set to detect changes to the memory location that is being corrupted. Alternatively, the memory location can be manually inspected by single stepping through the program. After setting the watchpoint, execution of the program is continued until the watchpoint is triggered, which occurs upon a change to the corrupted memory location, thereby breaking execution of the program. At each of the breaks in execution due to encountering the watchpoints, the programmer can use the debugger to inspect the program and data in search of the cause of the memory corruption. If the programmer does indeed concluded that the

latest watchpoint detected the actual corruption, rather than an intended use of the memory location, then the programmer can tell from the current program counter in the debugger which statement caused the corruption.").

***Claim Rejections - 35 USC § 103***

**10. Claim 5 rejected under 35 U.S.C. 103(a) as being unpatentable over US 6769116 to Sexton as applied to claim 1 above, and further in view of US 6052524 to Pauna.**

11. Referring to claim 5, although Sexton does not specifically disclose that said one or more simulation outputs comprise information from one or more of a processor simulator and one or more processor simulation models, using a hardware simulator in conjunction with software debugging is known in the art. An example of this is shown by Pauna, from the abstract, "A system and methods are provided to design, verify and develop simulated hardware and software components for a desired electrical device." A person of ordinary skill in the art at the time of the invention would have been motivated to perform hardware software co-simulation because from line 6 of column 2 of Pauna, "In today's world, "co-verification" using simulation technologies allows developers to design and verify the behavior of hardware and software components in a new electronic device. Behavior of hardware and software components can be examined during a simulation by setting break-points, stopping the simulation at various states, and examining data generated by the simulation. Various simulations also allow

certain hardware components to be emulated with a hardware description language while other hardware and software components are simulated."

**12. Claims 6, 7 rejected under 35 U.S.C. 103(a) as being unpatentable over US 6769116 to Sexton as applied to claim 1 above, and further in view of "frames" and "windows".**

13. Referring to claims 6, 7, although Sexton does not specifically disclose said graphical user interface is configured to present said debug information in one or more windows or frames, these are very well known in the art. Examiner takes official notice4 for "windows" and "frames". A person of ordinary skill in the art at the time of the invention would have been motivated to use windows or frames to display data because windows and frames, particularly in a GUI environment serve to graphically partition data for display to a user.

**14. Claim 10, 12, 14-19 rejected under 35 U.S.C. 103(a) as being unpatentable over US 6769116 to Sexton in view of US 6052524 to Pauna.**

15. Referring to claim 10, Sexton discloses (A) identifying a memory address to be examined (From line 9 of column 4, "To use these instrumented memory management routines, a programmer ascertains the address of the object that was corrupted, typically by inspecting the core dump, and sets the global pointer to that address." From line 15 of column 8, "To debug memory corruption, a predetermined pointer is set to the desired address, which is checked in the instrumented memory management routines, and a breakpoint is set on the breakpointable function. Thus, the execution of large portions of the program can be quickly passed over before starting the intensive

watchpointing or other memory inspection procedure.");

(B) retrieving one or more accesses related to said memory address; (C) identifying a specific one of said one or more accesses to be examined (From line 10 of column 9, "When the breakpoint is triggered (e.g. when a memory management routine handles an object with the address set in the global debug pointer), the debugger breaks execution of the problem, allowing the programmer to issue debugging commands to inspect memory locations and continue the program's running from the breakpoint. For example, when the memory allocation routine allocates memory for an object that starts with the address set in the global debug pointer, then the debugging routine would be called, triggering the breakpoint and suspending execution of the program within the debugger."); and

(D) retrieving one or more types of debug information related to said identified access (From line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." From line 30 of column 9, "At block 208, e.g. after the last breakpoint is triggered, a watchpoint is set to detect changes to the memory location that is being corrupted. Alternatively, the memory location can be manually inspected by single

stepping through the program. After setting the watchpoint, execution of the program is continued until the watchpoint is triggered, which occurs upon a change to the corrupted memory location, thereby breaking execution of the program. At each of the breaks in execution due to encountering the watchpoints, the programmer can use the debugger to inspect the program and data in search of the cause of the memory corruption. If the programmer does indeed concluded that the latest watchpoint detected the actual corruption, rather than an intended use of the memory location, then the programmer can tell from the current program counter in the debugger which statement caused the corruption.").

Although Sexton does not specifically disclose debugging RTL simulations of processor based system on chip (SoC), using a SoC simulator in conjunction with software debugging is known in the art. An example of this is shown by Pauna, from the abstract, "A system and methods are provided to design, verify and develop simulated hardware and software components for a desired electrical device." Further, from line 52 of column 1 of Pauna, "As electronic device development became increasing complex, very few hardware components of the system were built without performing a design verification first. The design verification was used to confirm that hardware components result in a design that operates in accordance with predetermined functional specifications. Hardware Description Languages ("HDLs") were developed to emulate and verify the design of hardware components. Hardware description languages known in the art include Very High Speed Integrated Circuit Hardware Description Language ("VHSIC HDL" or "VHDL")), Programming Control Language

("PCL") and others. Hardware description languages provide a gate-level emulation of hardware devices. For more information on VHDL, see Institute of Electrical and Electronic Engineers ("IEEE") standard 1076, which is incorporated herein by reference. Hardware description languages generally include a set of instructions for sending and receiving data to an emulated device under test, storing and receiving data, and other interactions. This allows the behavior of a proposed hardware device to be verified before it is built." Further, from line 50 of column 4 (with emphasis), "The system further includes a simulator library for modeling and verifying hardware components of a desired electronic device. The simulator library may include built-in models for simulating multiple internal and external hardware components (e.g., central processing units, **memory, memory management units, caches, timers**, universal asynchronous receiver transmitters and digital signal processors). The built-in models return a number of cycles on the **cycle-accurate** simulator executed for a desired simulated operation. The simulator library may also include simulator interface routines for setting a clock for a simulated component to a new clock speed, coordinating between a simulator library clock and a cycle-accurate simulator clock, handling events that occur before or during a current clock cycle, changing interrupt vectors and interrupt priority levels, providing **notification of changes in registers during a simulated operation**, or for setting one or more individual sub-components (e.g., status bits) of a simulated hardware component. The simulator library with built-in models and routines is used as an interface to the cycle-accurate simulator." A person of ordinary skill in the art at the time of the invention would have been motivated to perform hardware software co-simulation

because from line 6 of column 2 of Pauna, "In today's world, "co-verification" using simulation technologies allows developers to design and verify the behavior of hardware and software components in a new electronic device. Behavior of hardware and software components can be examined during a simulation by setting break-points, stopping the simulation at various states, and examining data generated by the simulation. Various simulations also allow certain hardware components to be emulated with a hardware description language while other hardware and software components are simulated."

16. Referring to claim 12, Sexton in view of Pauna discloses said one or more types of debug information comprise a register status of said processor (From line 65 of column 4 of Pauna, "providing notification of changes in registers during a simulated operation". Sexton, from line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." Sexton, from line 30 of column 9, "At block 208, e.g. after the last breakpoint is triggered, a watchpoint is set to detect changes to the memory location that is being corrupted. Alternatively, the memory location can be manually inspected by single stepping through the program. After setting the watchpoint, execution of the program is

continued until the watchpoint is triggered, which occurs upon a change to the corrupted memory location, thereby breaking execution of the program. At each of the breaks in execution due to encountering the watchpoints, the programmer can use the debugger to inspect the program and data in search of the cause of the memory corruption. If the programmer does indeed concluded that the latest watchpoint detected the actual corruption, rather than an intended use of the memory location, then the programmer can tell from the current program counter in the debugger which statement caused the corruption.").

17. Referring to claim 14, Sexton discloses said one or more types of debug information comprise a program structure (From line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." From line 30 of column 9, "At block 208, e.g. after the last breakpoint is triggered, a watchpoint is set to detect changes to the memory location that is being corrupted. Alternatively, the memory location can be manually inspected by single stepping through the program. After setting the watchpoint, execution of the program is continued until the watchpoint is triggered, which occurs upon a change to the corrupted memory location, thereby breaking execution of the

program. At each of the breaks in execution due to encountering the watchpoints, the programmer can use the debugger to inspect the program and data in search of the cause of the memory corruption. If the programmer does indeed concluded that the latest watchpoint detected the actual corruption, rather than an intended use of the memory location, then the programmer can tell from the current program counter in the debugger which statement caused the corruption.").

18. Referring to claim 15, Sexton discloses retrieving one or more accesses related to said memory address comprises: responding to activation of a button presented by a graphic user interface (From line 55 of column 5, "Computer system 100 may be coupled via bus 102 to a display 112, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 114, including alphanumeric and other keys, is coupled to bus 102 for communicating information and command selections to processor 104. Another type of user input device is cursor control 116, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 104 and for controlling cursor movement on display 112. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane." From line 55 of column 8, "When the instrumented code have been compiled into memory management or other library routines of the program under a debugging switch, the programmer is now ready to diagnose the cause of the memory corruption. Preferably, the following steps are performed within a symbolic debugger such as sdb (a standard UNIX.TM. debugger) or gdb (a freeware debugger), but some of the steps

may optionally be performed by hard coding some additional code into the program and recompiling.") configured (i) to present said command input in response to one or more user input parameters and (ii) to display said debug information (From line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." From line 10 of column 9, "When the breakpoint is triggered (e.g. when a memory management routine handles an object with the address set in the global debug pointer), the debugger breaks execution of the problem, allowing the programmer to issue debugging commands to inspect memory locations and continue the program's running from the breakpoint. For example, when the memory allocation routine allocates memory for an object that starts with the address set in the global debug pointer, then the debugging routine would be called, triggering the breakpoint and suspending execution of the program within the debugger.").

19. Referring to claim 16, Sexton discloses identifying a specific access is performed via a graphic user interface (From line 55 of column 5, "Computer system 100 may be coupled via bus 102 to a display 112, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 114, including alphanumeric and other

keys, is coupled to bus 102 for communicating information and command selections to processor 104. Another type of user input device is cursor control 116, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 104 and for controlling cursor movement on display 112. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane." From line 55 of column 8, "When the instrumented code have been compiled into memory management or other library routines of the program under a debugging switch, the programmer is now ready to diagnose the cause of the memory corruption. Preferably, the following steps are performed within a symbolic debugger such as sdb (a standard UNIX.TM. debugger) or gdb (a freeware debugger), but some of the steps may optionally be performed by hard coding some additional code into the program and recompiling.") configured (i) to present said command input in response to one or more user input parameters and (ii) to display said debug information (From line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." From line 10 of column 9, "When the breakpoint is triggered (e.g. when a memory management routine handles

an object with the address set in the global debug pointer), the debugger breaks execution of the problem, allowing the programmer to issue debugging commands to inspect memory locations and continue the program's running from the breakpoint. For example, when the memory allocation routine allocates memory for an object that starts with the address set in the global debug pointer, then the debugging routine would be called, triggering the breakpoint and suspending execution of the program within the debugger." From line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." From line 30 of column 9, "At block 208, e.g. after the last breakpoint is triggered, a watchpoint is set to detect changes to the memory location that is being corrupted. Alternatively, the memory location can be manually inspected by single stepping through the program. After setting the watchpoint, execution of the program is continued until the watchpoint is triggered, which occurs upon a change to the corrupted memory location, thereby breaking execution of the program. At each of the breaks in execution due to encountering the watchpoints, the programmer can use the debugger to inspect the program and data in search of the cause of the memory corruption. If the programmer does indeed concluded that the latest watchpoint detected the actual

corruption, rather than an intended use of the memory location, then the programmer can tell from the current program counter in the debugger which statement caused the corruption.").

20. Referring to claim 17, Sexton discloses retrieving one or more types of debug information is performed in response to a button of a graphic user interface being actuated (From line 55 of column 5, "Computer system 100 may be coupled via bus 102 to a display 112, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 114, including alphanumeric and other keys, is coupled to bus 102 for communicating information and command selections to processor 104. Another type of user input device is cursor control 116, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 104 and for controlling cursor movement on display 112. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane." From line 55 of column 8, "When the instrumented code have been compiled into memory management or other library routines of the program under a debugging switch, the programmer is now ready to diagnose the cause of the memory corruption. Preferably, the following steps are performed within a symbolic debugger such as sdb (a standard UNIX.TM. debugger) or gdb (a freeware debugger), but some of the steps may optionally be performed by hard coding some additional code into the program and recompiling.") configured (i) to present said command input in response to one or more user input parameters and (ii) to display said debug information (From line 31 of column 8, "If

equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." From line 10 of column 9, "When the breakpoint is triggered (e.g. when a memory management routine handles an object with the address set in the global debug pointer), the debugger breaks execution of the problem, allowing the programmer to issue debugging commands to inspect memory locations and continue the program's running from the breakpoint. For example, when the memory allocation routine allocates memory for an object that starts with the address set in the global debug pointer, then the debugging routine would be called, triggering the breakpoint and suspending execution of the program within the debugger." From line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." From line 30 of column 9, "At block 208, e.g. after the last breakpoint is

triggered, a watchpoint is set to detect changes to the memory location that is being corrupted. Alternatively, the memory location can be manually inspected by single stepping through the program. After setting the watchpoint, execution of the program is continued until the watchpoint is triggered, which occurs upon a change to the corrupted memory location, thereby breaking execution of the program. At each of the breaks in execution due to encountering the watchpoints, the programmer can use the debugger to inspect the program and data in search of the cause of the memory corruption. If the programmer does indeed concluded that the latest watchpoint detected the actual corruption, rather than an intended use of the memory location, then the programmer can tell from the current program counter in the debugger which statement caused the corruption.").

21. Referring to claim 18, Sexton discloses displaying said retrieved one or more accesses and said retrieved one or more types of debug information using a graphic user interface (From line 55 of column 5, "Computer system 100 may be coupled via bus 102 to a display 112, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 114, including alphanumeric and other keys, is coupled to bus 102 for communicating information and command selections to processor 104. Another type of user input device is cursor control 116, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 104 and for controlling cursor movement on display 112. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a

plane." From line 55 of column 8, "When the instrumented code have been compiled into memory management or other library routines of the program under a debugging switch, the programmer is now ready to diagnose the cause of the memory corruption. Preferably, the following steps are performed within a symbolic debugger such as sdb (a standard UNIX.TM. debugger) or gdb (a freeware debugger), but some of the steps may optionally be performed by hard coding some additional code into the program and recompiling.") configured (i) to present said command input in response to one or more user input parameters and (ii) to display said debug information (From line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." From line 10 of column 9, "When the breakpoint is triggered (e.g. when a memory management routine handles an object with the address set in the global debug pointer), the debugger breaks execution of the problem, allowing the programmer to issue debugging commands to inspect memory locations and continue the program's running from the breakpoint. For example, when the memory allocation routine allocates memory for an object that starts with the address set in the global debug pointer, then the debugging routine would be called, triggering the breakpoint and suspending execution of the program within the

debugger." From line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." From line 30 of column 9, "At block 208, e.g. after the last breakpoint is triggered, a watchpoint is set to detect changes to the memory location that is being corrupted. Alternatively, the memory location can be manually inspected by single stepping through the program. After setting the watchpoint, execution of the program is continued until the watchpoint is triggered, which occurs upon a change to the corrupted memory location, thereby breaking execution of the program. At each of the breaks in execution due to encountering the watchpoints, the programmer can use the debugger to inspect the program and data in search of the cause of the memory corruption. If the programmer does indeed concluded that the latest watchpoint detected the actual corruption, rather than an intended use of the memory location, then the programmer can tell from the current program counter in the debugger which statement caused the corruption.").

22. Referring to claim 19, Sexton discloses modifying information displayed using said graphic user interface in response to one or more filter commands (From line 55 of column 5, "Computer system 100 may be coupled via bus 102 to a display 112, such as

a cathode ray tube (CRT), for displaying information to a computer user. An input device 114, including alphanumeric and other keys, is coupled to bus 102 for communicating information and command selections to processor 104. Another type of user input device is cursor control 116, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 104 and for controlling cursor movement on display 112. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane." From line 55 of column 8, "When the instrumented code have been compiled into memory management or other library routines of the program under a debugging switch, the programmer is now ready to diagnose the cause of the memory corruption. Preferably, the following steps are performed within a symbolic debugger such as sdb (a standard UNIX.TM. debugger) or gdb (a freeware debugger), but some of the steps may optionally be performed by hard coding some additional code into the program and recompiling.") configured (i) to present said command input in response to one or more user input parameters and (ii) to display said debug information (From line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the

memory management routines." From line 10 of column 9, "When the breakpoint is triggered (e.g. when a memory management routine handles an object with the address set in the global debug pointer), the debugger breaks execution of the problem, allowing the programmer to issue debugging commands to inspect memory locations and continue the program's running from the breakpoint. For example, when the memory allocation routine allocates memory for an object that starts with the address set in the global debug pointer, then the debugging routine would be called, triggering the breakpoint and suspending execution of the program within the debugger." From line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." From line 30 of column 9, "At block 208, e.g. after the last breakpoint is triggered, a watchpoint is set to detect changes to the memory location that is being corrupted. Alternatively, the memory location can be manually inspected by single stepping through the program. After setting the watchpoint, execution of the program is continued until the watchpoint is triggered, which occurs upon a change to the corrupted memory location, thereby breaking execution of the program. At each of the breaks in execution due to encountering the watchpoints, the programmer can use the debugger to inspect the

program and data in search of the cause of the memory corruption. If the programmer does indeed concluded that the latest watchpoint detected the actual corruption, rather than an intended use of the memory location, then the programmer can tell from the current program counter in the debugger which statement caused the corruption.").

**23. Claim 11, 13 rejected under 35 U.S.C. 103(a) as being unpatentable over US 6769116 to Sexton in view of 6052524 to Pauna as applied to claim 10 above, and further in view of "assembly code".**

24. Referring to claim 11, Sexton discloses said one or more types of debug information comprise one or more instruction codes (From line 30 of column 9 (with emphasis), "At block 208, e.g. after the last breakpoint is triggered, a watchpoint is set to detect changes to the memory location that is being corrupted. Alternatively, the memory location can be manually inspected by **single stepping through the program**. After setting the watchpoint, execution of the **program** is continued until the watchpoint is triggered, which occurs upon a change to the corrupted memory location, thereby breaking execution of the program. At each of the breaks in execution due to encountering the watchpoints, the programmer can use the debugger to **inspect the program and data** in search of the cause of the memory corruption. If the programmer does indeed concluded that the latest watchpoint detected the actual corruption, rather than an intended use of the memory location, then the programmer can tell from the current program counter in the debugger **which statement** caused the corruption.").

Although Sexton does not specifically disclose this instruction code may be assembler instruction code, assembly code is very well known in the art. Examiner

takes official notice for "assembly code". A person of ordinary skill in the art at the time of the invention would have been motivated to use assembly because it is faster than programming in machine code and provides efficient control over processor operations.

25. Referring to claim 13, Sexton discloses said one or more types of debug information comprise a program flow leading to said one or more assembler instruction codes (Sexton, from line 31 of column 8, "If equal, the memory allocation routine would then call a debugging function. The debugging function need not do anything at all and can be a stub function that merely returns back to the calling function as long as the debugging function allows for a breakpoint to be set in the debugger. In some embodiments, however, the debugging function can also output useful information, such as the address, type, and value of the object to the screen or to a log file. In addition, a separate debugging routine may be provided for each of the memory management routines." Sexton, from line 30 of column 9, "At block 208, e.g. after the last breakpoint is triggered, a watchpoint is set to detect changes to the memory location that is being corrupted. Alternatively, the memory location can be manually inspected by single stepping through the program. After setting the watchpoint, execution of the program is continued until the watchpoint is triggered, which occurs upon a change to the corrupted memory location, thereby breaking execution of the program. At each of the breaks in execution due to encountering the watchpoints, the programmer can use the debugger to inspect the program and data in search of the cause of the memory corruption. If the programmer does indeed concluded that the latest watchpoint detected the actual corruption, rather than an intended use of the memory location, then the programmer

can tell from the current program counter in the debugger which statement caused the corruption.").

***Conclusion***

26. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure. See notice of references cited.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Gabriel L. Chu whose telephone number is (571) 272-3656. The examiner can normally be reached on weekdays between 8:30 AM and 5:00 PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Scott Baderman can be reached on (571) 272-3644. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

Gabriel L. Chu
Primary Examiner
Art Unit 2114

gc